# Appendix A: Theoretical Concepts

## *Baseline Algorithm – kNN (k Nearest Neighbors)*

In order to compare our neural network performance, we have built a kNN classifier. This classifier builds a model from training images by arranging them in a d-dimensional space by some distance measure. In order to classify a test image, it then finds k training images, which are closest to a test image and uses most frequent label as the solution [12].

Each image in the set is represented by a feature vector (FV). FV in the simplest form could be a vector of unrolled grayscale image pixel values. For an image with dimensions, the FV is.

The performance of kNN classifier on raw data is likely to be very poor. In order to achieve better classification accuracy of the kNN classifier, two different dimensionality reduction and feature extraction techniques can be used:

1. Principal Component Analysis (PCA) - Aims to explain variance in data by transforming the data to a new set of uncorrelated features, the principal components (PCs) [13]. PCs have much lower dimensionality and preserve all of the original variance. They are ranked (descending order) by the amount of variance they explain, so removing last n PCs, most likely noise and unimportant data is scrapped.
2. Linear Discriminant Analysis (LDA) – a feature extraction technique originally developed by R. A. Fisher in 1936. The algorithm is based on searching for a linear combination of variables that best separate two classes. A generalized version of LDA [14] has been used, which allows working with multiple classes. LDA is similar to PCA in the sense, that they both try to extract linear combinations of variables which best explain the data [15]. However LDA attempts to model differences between different classes of data, which seems to be a perfect tool for trying to distinguish between faces of different whales.

## *Backpropagation*

The backpropagation algorithm is used while training to propagate the error corrections from the output layer down through all the connected layers of neurons. This will ensure that the output result on the next epoch of training will provide a better score.

The main idea of the algorithm is using gradient descent to calculate the required changes of the current weights to reach a value with a minimal error difference from the required output. If the output vector was defined to be a dot product of the weights vector and the input vector the following equation would be found:

The training error of this linear unit (preceptor without the thresholding function) can be calculated by many functions – a simple way is summing the squared difference (t – refers to ideal output):

It is important to realize the correspondence of the weights to the training error will be some sort of hyperparabola – the specific shape of which will of course depend on the dataset. Here is an image of one such imagined parabola with only two weights. The global minima of the parabola will correspond to the best possible weights.



Figure 1: Possible paraboloid graph of neuron errors w.r.t. the weight values

Taking the gradient of the error vector with respect to each of the weights:

If one were to evaluate with the current weights – the result would be a vector showing the steepest increase of slope away from the current point. The negative of that would be the steepest decrease. Therefore new weights can be updated with the following delta:

is the learning rate. Setting to a greater value can result in a quicker convergence, however it can overshoot the global minima, while setting it too low will result in a very slow training process.

Gradient descent is not guaranteed to reach the global minima, however this is less of a problem when N is large for N-dimensional data as there will be more “routes” of sliding down towards the minima with a larger number of weights. Backpropagation is simply applying gradient descent recursively from the outermost layer all the way through the network.

## *Adam Optimizer*

An improvement on the classical gradient descent algorithm would be to maintain inertia while descending towards the minima. The simple gradient descent algorithm can be prone to radically changing directions of descent if the data is noisy. 1st and 2nd order moments can be added to increase the chance of staying on the true path while training.

The Adam (adaptive moment estimation) algorithm is described by (Kingma & Ba, 2014). It is a state of the art method for training. It utilizes two extra variables while calculating the weight updates:

– exponential moving average of the gradient ()

– exponential moving average of the elementwise squared gradient ()

On the first iteration both of the vectors are set to zeroes. After wards they will be calculated using their corresponding exponential decay rates and epsilon .

On each iteration of the Adam gradient descent first the biased moment estimates are found:

The next step is to compute bias corrected moment estimates by dividing by one minus β to the power of t:

Finally the new weights can be recalculated:

## *Soft-max activation for output neurons*

The output layer of neurons is used for classification, however all of the neurons may have different activation levels. In order to estimate the activations as probabilities () we need to make sure that all of the outputs sum to one. This can be achieved with the soft-max activation function:

## *Regularization*

In order to prevent overfitting we used dropout of neurons with 0.5 probability of dropout in the fully connected layers. The activation levels of the convolutional layers were regularized with L2 normalization or with local response normalization outlined by Krizhevsky et al [1].